



TITLE:

# An Improved Randomized Approximation Algorithm for Max TSP (Theoretical Computer Science and its Applications)

AUTHOR(S):

Chen, Zhi-Zhong; Wang, Lusheng

---

CITATION:

Chen, Zhi-Zhong ...[et al]. An Improved Randomized Approximation Algorithm for Max TSP (Theoretical Computer Science and its Applications). 数理解析研究所講究録 2005, 1426: 7-12

ISSUE DATE:

2005-04

URL:

<http://hdl.handle.net/2433/47267>

RIGHT:

# An Improved Randomized Approximation Algorithm for Max TSP

Zhi-Zhong Chen (陳 致中)  
Department of Mathematical Sciences  
Tokyo Denki University  
Hatoyama, Saitama 350-0394, Japan.  
(東京電機大学理工学部数理科学科)

Lusheng Wang (王 魯生)  
Department of Computer Science  
City University of Hong Kong  
Tat Chee Avenue, Kowloon, Hong Kong.  
(香港城市大学計算機系)

## Abstract

We present an  $O(n^3)$ -time randomized approximation algorithm for the maximum traveling salesman problem whose expected approximation ratio is asymptotically  $\frac{251}{331}$ , where  $n$  is the number of vertices in the input (undirected) graph. This improves the previous best.

## 1 Introduction

The *maximum traveling salesman problem* (Max TSP) is to compute a maximum-weight Hamiltonian circuit (called a *tour*) in a given edge-weighted (undirected) graph. The problem is known to be Max-SNP-hard [1] and there have been a number of approximation algorithms known for it [3, 4, 7]. In 1984, Serdyukov [7] gave an  $O(n^3)$ -time approximation algorithm for Max TSP that achieves an approximation ratio of  $\frac{3}{4}$ . Serdyukov's algorithm is very simple and elegant, and it tempts one to ask if a better approximation ratio can be achieved for Max TSP by a polynomial-time approximation algorithm. Along this line, Hassin and Rubinstein [4] showed that with the help of randomization, better approximation ratio for Max TSP can be achieved. More precisely, they gave an  $O(n^3)$ -time randomized approximation algorithm for Max TSP whose *expected* approximation ratio is asymptotically  $\frac{25}{33}$ . Their algorithm is basically a combination of Serdyukov's algorithm and an earlier algorithm of their own [3].

The asymptotic ratio  $\frac{25}{33}$  achieved by Hassin and Rubinstein's algorithm is marginally better than the ratio  $\frac{3}{4}$  achieved by Serdyukov's algorithm. However, Hassin and Rubinstein said in their paper [4]: "the better ratio at least demonstrates that the ratio of  $\frac{3}{4}$  can be improved and further research along this line is encouraged". Moreover, it is widely recognized that improving approximation algorithms for TSP and its variants are not easy. In this paper, following and improving Hassin and Rubinstein's work, we give a new  $O(n^3)$ -time randomized approximation algorithm for Max TSP whose expected approximation ratio is asymptotically  $\frac{251}{331}$ . Hassin and Rubinstein [4] show that each approximation algorithm for Max TSP can be translated into an approximation algorithm for a problem called the *maximum latency TSP* which was first studied by Chalasani and Motwani [2]. Using their translation, our new algorithm can be trivially turned into a new randomized approximation algorithm for the maximum latency TSP whose expected approximation ratio improves the previous best.

Like all previous approximation algorithms for Max TSP, our new algorithm starts by computing a maximum-weight cycle cover  $\mathcal{C}$  of the input graph  $G$  and then modify the cycles in  $\mathcal{C}$  (somehow) to a tour of  $G$  without losing much weight. All the previous algorithms modify the cycles in  $\mathcal{C}$  in an arbitrary order. In contrast, our algorithm modify the cycles in a carefully chosen order based on suitably constructed auxiliary graphs. Moreover, the way of modifying a

cycle heavily depends on how the previous cycles were modified. This is why our algorithm is complicated.

Throughout the rest of the paper, fix an instance  $(G, w)$  of Max TSP, where  $G$  is a complete (undirected) graph and  $w$  is a function mapping each edge  $e$  of  $G$  to a nonnegative real number  $w(e)$ . For a subset  $F$  of  $E(G)$ ,  $w(F)$  denotes  $\sum_{e \in F} w(e)$ . The *weight* of a subgraph  $H$  of  $G$  is  $w(H) = w(E(H))$ . Our goal is to compute a tour of large weight in  $G$ . For ease of explanation, we assume that  $n = |V(G)|$  is even.

We first sketch our randomized algorithm for Max TSP in the next section and then describe its details in Sections 3 through 5. For a random variable  $X$ ,  $\mathcal{E}[X]$  denotes its expected value. For a random event  $A$ ,  $\Pr[A]$  denotes the probability that  $A$  occurs.

## 2 Outline of the New Randomized Algorithm

Like Hassin and Rubinstein's randomized algorithm (H&R-algorithm) for Max TSP, our algorithm starts by computing a maximum-weight cycle cover  $\mathcal{C}$  of  $G$ , uses it to compute three tours  $T_1, \dots, T_3$  of  $G$ , and outputs the one of the largest weight among them. Our computation of  $T_1$  is the same as in H&R-algorithm. Our computation of  $T_2$  and  $T_3$  is as shown in Figure 1:

1. Compute a maximum-weight matching  $M$  in  $G$ , and compute a maximum-weight matching  $M'$  in a graph  $H$ , where  $V(H) = V(G)$  and  $E(H)$  consists of those  $\{u, v\} \in E(G)$  such that  $u$  and  $v$  belong to different cycles in  $\mathcal{C}$ . (Note: Since  $|V(G)|$  is even,  $M$  is perfect.)
2. Let  $C_1, \dots, C_r$  be an ordering of the cycles in  $\mathcal{C}$  such that  $C_1, \dots, C_\ell$  are the 4-cycles in  $\mathcal{C}$ .
3. Make a backup copy  $M_c$  of  $M$ .
4. Process  $C_1, \dots, C_\ell$  in a suitable order, by (1) coloring some edges  $\{u, v\} \in M'$  with  $\{u, v\} \subseteq \cup_{1 \leq i \leq \ell} V(C_i)$  *red*, and (2) moving exactly one suitable edge from each 4-cycle to  $M$  while always maintaining that the graph  $(V(G), M)$  is a subtour of  $G$ .
5. Process  $C_{\ell+1}, \dots, C_r$  one by one in this order, by (1) coloring some edges  $\{u, v\} \in M'$  with  $\{u, v\} \cap (\cup_{\ell+1 \leq i \leq r} V(C_i)) \neq \emptyset$  *red* or *green*, and (2) moving one or more suitable edges in each non-4-cycle to  $M$  while always maintaining that graph  $(V(G), M)$  is a subtour of  $G$ .
6. Add to  $\mathcal{C}$  those edges  $\{u, v\} \in M' - R$  such that both  $u$  and  $v$  have degree 1 in  $\mathcal{C}$ , where  $R$  is the set of *red* edges in  $M'$ . (Note: Let  $M'_\delta$  denote the set of edges in  $M'$  that are added to  $\mathcal{C}$  at this step. Immediately after this step,  $|E(C) \cap M'_\delta| \geq 2$  for each cycle  $C$  in  $\mathcal{C}$ .)
7. For each cycle  $C$  in  $\mathcal{C}$ , if  $|E(C) \cap M'| = 2$  and one edge in  $E(C) \cap M'$  is *green*, then delete one edge in  $E(C) \cap M'$  from  $\mathcal{C}$  at random in such a way that the *green* edge is deleted with probability  $2/3$ ; otherwise, select one edge in  $E(C) \cap M'$  uniformly at random and delete it from  $\mathcal{C}$ . (Note: Let  $M'_\gamma$  denote the set of edges in  $M'$  that remain in  $\mathcal{C}$  immediately after this step.)
8. Complete  $\mathcal{C}$  to a tour  $T_2$  of  $G$  by adding some edges of  $G$ , and complete the graph  $(V(G), M)$  to a tour  $T_3$  of  $G$  by adding some edges of  $G$ .

Figure 1. Computation of  $T_2$  and  $T_3$  in our algorithm. (Steps 4 and 5 are rough.)

Steps 4 and 5 in Figure 1 are rough; their details are very complicated and will be given in the subsequent sections. An important property will be that  $w(R)$  is small compared with  $w(M')$ .

Several definitions and two useful facts are in order. Throughout the rest of this paper, for each integer  $i \in \{1, \dots, r\}$ , the phrase “at time  $i$ ” means the time at which zero or more cycles in  $\mathcal{C}$  have been processed and  $C_i$  is the next cycle to be processed. A set  $F$  of edges in  $G$  is *available at time  $i$*  if  $F$  is a matching in  $C_i$ ,  $F \cap M_c = \emptyset$ , and the graph  $(V(G), M \cup F)$  is a subtour of  $G$  at time  $i$ . An edge  $e$  in  $G$  is *available at time  $i$*  if  $\{e\}$  is available at time  $i$ . A *maximal available set at time  $i$*  is an available set  $F$  at time  $i$  such that for every  $e \in E(C_i) - F$ ,  $F \cup \{e\}$  is not available at time  $i$ .

**Lemma 2.1** *Let  $F$  be an available set at time  $i$ . Suppose that  $e_1 = \{u_1, u_2\}$  and  $e_2 = \{u_2, u_3\}$  are two adjacent edges in  $C_i$  such that  $F$  contains no edge incident to  $u_1$ ,  $u_2$ , or  $u_3$ . Then,  $F \cup \{e_1\}$  or  $F \cup \{e_2\}$  is available at time  $i$ .*

### 3 Processing 4-Cycles

We say that two distinct edges  $e_1 = \{u_1, v_1\}$  and  $e_2 = \{u_2, v_2\}$  in  $M'$  form a *square pair*, denoted by  $\{e_1, e_2\}_{\text{sp}}$ , if  $\{u_1, u_2\}$  is an edge in a 4-cycle  $C_i$  and  $\{v_1, v_2\}$  is an edge in another 4-cycle  $C_j$ . We call  $C_i$  and  $C_j$  the *dependent 4-cycles* of the square pair. An edge  $e \in M'$  is a *square edge* if  $e$  is contained in some square pair.

We construct a multigraph  $H_1$  from  $M'$  and  $C_1, \dots, C_\ell$  as follows. The nodes of  $H_1$  one-to-one correspond to  $C_1, \dots, C_\ell$ . For convenience, we still use  $C_i$  ( $1 \leq i \leq \ell$ ) to denote the node of  $H_1$  corresponding to it. The edges of  $H_1$  one-to-one correspond to the square pairs. In more detail, corresponding to each square pair  $p$ ,  $H_1$  has an edge between the dependent 4-cycles of  $p$ .  $H_1$  has no other edges. For each edge  $f$  of  $H_1$ , we denote the square pair corresponding to  $f$  by  $p(f)$ .

An edge  $\{u, v\} \in M'$  is *4-cycle-closed* if there are two 4-cycles  $C_i$  and  $C_j$  in  $\mathcal{C}$  with  $u \in V(C_i)$  and  $v \in V(C_j)$ . An edge  $e \in M'$  is *4-cycle-pendent* if for exactly one endpoint  $u$  of  $e$ , there is a 4-cycle  $C_i$  in  $\mathcal{C}$  with  $u \in V(C_i)$ . Let  $Q$  be a connected subgraph of  $H_1$ . An edge  $\{u, v\} \in M'$  is *Q-closed* if there are two nodes  $C_i$  and  $C_j$  in  $Q$  with  $u \in V(C_i)$  and  $v \in V(C_j)$ . An edge  $e \in M'$  is *Q-pendent* if for exactly one endpoint  $u$  of  $e$ , there is a node  $C_i$  in  $Q$  with  $u \in V(C_i)$ . The *weight* of  $Q$  is the total weight of  $Q$ -closed edges in  $M'$ , and is denoted by  $w(Q)$ .

Obviously, we can classify the connected components  $Q$  of  $H_1$  into ten types as follows:

**Type 1:**  $Q$  is a single node.

**Type 2:**  $Q$  is a bunch of four parallel edges between two nodes.

**Type 3:**  $Q$  is an odd cycle.

**Type 4:**  $Q$  is an even cycle of length 4 or more.

**Type 5:**  $Q$  is a path of length 1 or more, and  $Q$  has an endpoint  $C_i$  (a 4-cycle in  $\mathcal{C}$ ) such that neither a  $Q$ -pendent edge nor a  $Q$ -closed non-square edge is incident to a vertex of  $C_i$ . (Note: We call  $C_i$  a *dead end* of  $Q$ . Note that if there is a  $Q$ -closed non-square edge, then  $Q$  has no dead end.)

**Type 6:**  $Q$  is a path of length 3 or more, and  $Q$  has no dead end.

**Type 7:**  $Q$  is a 2-cycle.

**Type 8:**  $Q$  is a path of length 1 and  $Q$  has no dead end.

**Type 9:**  $Q$  is a path of length 2,  $Q$  has no dead end, and there is no  $Q$ -closed non-square edge.

**Type 10:**  $Q$  is a path of length 2 and there is a  $Q$ -closed non-square edge.

**Lemma 3.1** *Suppose that our algorithm has processed zero or more 4-cycles and that  $C_i$  and  $C_j$  are two distinct 4-cycles not yet processed. Let  $e_1$  and  $e_2$  be two nonadjacent edges in  $C_i$  such that for each  $e_k \in \{e_1, e_2\}$ ,  $e_k \notin M_c$  and the graph  $(V(G), M \cup \{e_k\})$  is a subtour of  $G$ . Then, we can choose two nonadjacent edges  $e_3$  and  $e_4$  in  $E(C_j) - M_c$  such that for each  $e_x \in \{e_1, e_2\}$  and for each  $e_y \in \{e_3, e_4\}$ , the graph  $(V(G), M \cup \{e_x, e_y\})$  is a subtour of  $G$ .*

**Corollary 3.2** *For every 4-cycle  $C_i$  in  $\mathcal{C}$ , there are two nonadjacent edges available at time  $i$ .*

To process the 4-cycles in  $\mathcal{C}$ , our algorithm considers the connected components of  $H_1$  one by one. When considering a connected component  $Q$  of  $H_1$ , our algorithm processes those 4-cycles (in a row) that are nodes of  $Q$ . Since the details heavily depend on the type of  $Q$ , we describe the Type-5 case immediately and omit the details of the other cases for lack of space:

1. Let  $C_{i_1}$  and  $C_{i_2}$  be the endpoints of path  $Q$ , where node  $C_{i_2}$  is a dead end of  $Q$ . Let  $f_1 = \{C_{i_1}, C_{i_3}\}$  be the edge of  $Q$  incident to node  $C_{i_1}$ .
2. Let  $E_{i_1}$  be a set of two nonadjacent edges in  $E(C_{i_1}) - M_c$  such that for each  $e_x \in E_{i_1}$ , the graph  $(V(G), M \cup \{e_x\})$  is a subtour of  $G$ . (Note: By Corollary 3.2,  $E_{i_1}$  exists.)
3. Partition  $E(Q)$  into two disjoint matchings  $N_1$  and  $N_2$ .
4. Select an  $h \in \{1, 2\}$  uniformly at random.
5. If  $f_1 \in N_h$ , then perform the following steps:
  - (a) Select an  $e \in p(f_1)$  uniformly at random.
  - (b) Color  $e$  *purple*, and color the other edge in  $p(f_1)$  *red*.
  - (c) Move the edge in  $E_{i_1}$  adjacent to  $e$  from  $C_{i_1}$  to  $M$ .
  - (d) Find an edge  $e' \in E(C_{i_3}) - M_c$  adjacent to  $e$  such that the graph  $(V(G), M \cup \{e'\})$  is a subtour of  $G$ ; further move  $e'$  from  $C_{i_3}$  to  $M$ . (Note: By Corollary 3.2,  $e'$  exists.)
6. If  $f_1 \notin N_h$ , then perform the following step:
  - (a) If there is an edge  $e' \in E_{i_1}$  such that no edge in  $p(f_1)$  is adjacent to  $e'$ , then move  $e'$  from  $C_{i_1}$  to  $M$ ; otherwise, select an  $e'' \in E_{i_1}$  uniformly at random, and move  $e''$  from  $C_{i_1}$  to  $M$ .
7. If node  $C_{i_2}$  is incident to no edge in  $N_h$ , then move an edge  $e \in E(C_{i_2}) - M_c$  from  $C_{i_2}$  to  $M$  such that the graph  $(V(G), M \cup \{e\})$  is a subtour of  $G$ . (Note: By Corollary 3.2,  $e$  exists.)
8. For each edge  $f \in N_h - \{f_1\}$ , perform the following steps:
  - (a) Let  $C_i$  and  $C_j$  be the dependent 4-cycles of  $p(f)$ .
  - (b) Select an  $e \in p(f)$  uniformly at random.
  - (c) Color  $e$  *purple*, and color the other edge in  $p(f)$  *red*.
  - (d) Find an edge  $e' \in E(C_i) - M_c$  incident to an endpoint of  $e$  such that the graph  $(V(G), M \cup \{e'\})$  is a subtour of  $G$ ; further move  $e'$  from  $C_i$  to  $M$ .
  - (e) Find an edge  $e'' \in E(C_j) - M_c$  incident to an endpoint of  $e$  such that the graph  $(V(G), M \cup \{e''\})$  is a subtour of  $G$ ; further move  $e''$  from  $C_j$  to  $M$ .
9. Color all uncolored  $Q$ -closed edges *red*.

Figure 2. Steps for processing a Type-5 connected component  $Q$  of  $H_1$ .

In general, immediately after considering a connected component  $Q$  of  $H_1$  and processing the 4-cycle(s) that are nodes of  $Q$ , the following three invariants hold:

- (I1) The graph  $(V(G), M)$  remains to be a subtour of  $G$ .
- (I2) Let  $C_i$  be a 4-cycle that is a node of  $Q$ . Then, exactly one edge of  $C_i$  was moved from  $C_i$  to  $M$  during considering  $Q$ .
- (I3) Let  $u$  be a vertex in a 4-cycle  $C_i$  that is a node of  $Q$ . Suppose that no  $Q$ -closed edge in  $M'$  is incident to  $u$ . Then, with probability at least  $1/2$ , exactly one edge of  $C_i$  incident to  $u$  was moved from  $C_i$  to  $M$  during considering  $Q$ .

Obviously, immediately after considering a Type-5 connected component  $Q$  of  $H_1$ , Invariants (I1) through (I3) hold. We can show that this is also true after considering a connected component of each other type. Then, we can further show the following (main) lemma:

**Lemma 3.3** *Immediately after Step 4 in Figure 1 (i.e., immediately after processing the 4-cycles  $C_1, \dots, C_\ell$ ), the following hold:*

1. The graph  $(V(G), M)$  is a subtour of  $G$ .
2. Each  $C_i$  ( $1 \leq i \leq \ell$ ) becomes a path in  $C$ .
3. Let  $e$  be a 4-cycle-pendent edge in  $M'$ . Then, with probability at least  $1/2$ , the endpoint of  $e$  in a  $C_i$  ( $1 \leq i \leq \ell$ ) is of degree 1 in  $C$ .
4. Let  $S$  be the set of 4-cycle-closed edges in  $M'$ . Then,  $\mathcal{E}[w(S \cap M'_7)] \geq w(S)/6$ .

## 4 Processing Non-4-Cycles

For convenience, we transform each edge  $\{u, v\} \in M'$  to an ordered pair  $(u, v)$ , where the  $C_i$  with  $u \in V(C_i)$  and the  $C_j$  with  $v \in V(C_j)$  satisfy that  $i > j$ .

Let  $i$  be an integer in  $\{\ell + 1, \dots, r\}$ . A  $C_i$ -settled edge is an edge  $(u, v) \in M'$  such that  $u \in V(C_i)$  (and so  $v \in V(C_j)$  for some  $j < i$ ). A  $C_i$ -settled edge  $(u, v)$  is *active* at time  $i$  if the degree of  $v$  in  $C$  at time  $i$  is 1. A  $C_i$ -settled vertex is a vertex of  $C_i$  incident to a  $C_i$ -settled edge.

A *matching-pair* in  $C_i$  is an (unordered) pair  $\{A_1, A_2\}$  such that both  $A_1$  and  $A_2$  are (possibly empty) matchings in  $C_i$ . An *available matching-pair at time  $i$*  is a matching-pair  $\{A_1, A_2\}$  in  $C_i$  such that both  $A_1$  and  $A_2$  are available at time  $i$ . A *maximal available matching-pair at time  $i$*  is a matching-pair  $\{A_1, A_2\}$  in  $C_i$  such that both  $A_1$  and  $A_2$  are maximal available sets at time  $i$ .

A matching-pair  $\{A_1, A_2\}$  in  $C_i$  *covers* a vertex  $u$  of  $C_i$  if at least one edge in  $A_1 \cup A_2$  is incident to  $u$ . A matching-pair  $\{A_1, A_2\}$  in  $C_i$  *favors* a vertex  $u$  of  $C_i$  if  $A_1$  contains an edge  $e_1 \in E(C_i)$  incident to  $u$  and  $A_2$  contains an edge  $e_2 \in E(C_i)$  incident to  $u$  (possibly  $e_1 = e_2$ ). Figure 3 shows a procedure useful for computing an available matching-pair at time  $i$  that covers the vertices of a given subgraph  $P$  of  $C_i$ .

**Procedure FindMatch( $i, Y_1, Y_2, P, e$ )**

**Input:** An integer  $i \in \{\ell + 1, \dots, r\}$ ; an available matching-pair  $\{Y_1, Y_2\}$  at time  $i$  with  $Y_1 \cap Y_2 = \emptyset$ ; a subgraph  $P$  of  $C_i$  and an edge  $e$  of  $P$  such that  $|E(P)| \geq 2$ ,  $E(P) \cap Y_1 = E(P) \cap Y_2 = \emptyset$ ,  $Y_1 \cup \{e_1\}$  is available at time  $i$ , and either  $P = C_i$  or  $P$  is a path in  $C_i$  beginning with  $e$ .

1. Let  $e_1, \dots, e_t$  be the edges in  $P$  (appearing in  $P$  in this order) where  $e_1 = e$ . Let  $u_1$  be the endpoint of  $e_1$  not incident to  $e_2$ . Let  $u_2$  be the endpoint of  $e_t$  not incident to  $e_{t-1}$ .
2. Initialize  $Z_1 = Y_1$ ,  $Z_2 = Y_2$ , and  $j = h = 1$ .
3. While  $j < t$ , perform the following two steps:
  - (a) If  $Z_h \cup \{e_j\}$  is available at time  $i$ , then add  $e_j$  to  $Z_h$  and further increase  $j$  by 1; otherwise, add  $e_{j+1}$  to  $Z_h$  and further increase  $j$  by 2.
  - (b) If  $h = 1$ , then set  $h = 2$ ; otherwise, set  $h = 1$ .
4. If some  $Z_k$  with  $k \in \{1, 2\}$  contains both edges of  $C_i$  incident to  $u_2$ , then  $(e_{t-1} \notin Z_1 \cup Z_2$  and so) perform the following two steps:
  - (a) Let  $e'$  be the edge in  $E(C_i) - \{e_t\}$  incident to  $u_2$ . Let  $e''$  be the edge in  $E(C_i) - \{e_t, e'\}$  adjacent to  $e'$ .
  - (b) If  $e'' \in Z_1 \cup Z_2$ , then delete  $e'$  from  $Z_k$ .
  - (c) If  $e'' \notin Z_1 \cup Z_2$ , then move a suitable edge in  $\{e_t, e'\}$  from  $Z_k$  to  $Z_{k'}$  while maintaining that  $Z_{k'}$  is available at time  $i$ , where  $k'$  is the integer in  $\{1, 2\} - \{k\}$ .

**Output:** The ordered pair  $(Z_1, Z_2)$ .

Figure 3. A procedure useful for computing  $A_1$  and  $A_2$ .

#### 4.1 Serious Pairs, Critical Pairs, and Dangerous Pairs

Throughout this subsection, fix a  $C_i$  with  $\ell + 1 \leq i \leq r$ . A *serious pair at time  $i$*  is an unordered pair  $\{(u_1, v_1), (u_2, v_2)\}$  of  $C_i$ -settled edges satisfying the following condition:

- At time  $i$ , some connected component of  $C$  is a path between  $v_1$  and  $v_2$ .

A matching-pair  $\{A_1, A_2\}$  in  $C_i$  is *good* for a serious pair  $p = \{(u_1, v_1), (u_2, v_2)\}$  at time  $i$  if  $\{A_1, A_2\}$  satisfies at least one of the following three conditions:

- (G1) For each  $h \in \{1, 2\}$ ,  $C_i - A_h$  has no path from  $u_1$  to  $u_2$  or at least one of  $u_1$  and  $u_2$  has degree 2 in  $C_i - A_h$ .
- (G2)  $\{A_1, A_2\}$  favors both  $u_1$  and  $u_2$ .
- (G3)  $\{A_1, A_2\}$  favors exactly one of  $u_1$  and  $u_2$ . (Note: If this condition is satisfied but Condition (G1) is not, we say that  $\{A_1, A_2\}$  is *weakly good* for  $p$ .)

A *critical pair at time  $i$*  is a serious pair  $p = \{(u_1, v_1), (u_2, v_2)\}$  at time  $i$  such that there is a path  $Q$  from  $u_1$  to  $u_2$  in  $C_i$  with  $|E(Q)| \leq 3$ . We call the path  $Q$  a *witness path* of the critical pair  $p$ . A *dangerous pair at time  $i$*  is a critical pair  $p = \{(u_1, v_1), (u_2, v_2)\}$  at time  $i$  that has a witness path  $Q$  of length 1 or 3 satisfying the following condition:

- $\{e_1, e_2\}$  is an available set at time  $i$ , where  $e_1$  and  $e_2$  are the two edges in  $E(C_i) - E(Q)$  incident to an endpoint of  $Q$ .

## 4.2 Details of Processing Non-4-Cycles

If there is no dangerous pair at time  $i$ , then we color no vertex of  $C_i$  red and process  $C_i$  as below:

1. Find an available edge  $e$  at time  $i$ , and let  $(A_1, A_2)$  be the output of  $FindMatch(i, \emptyset, \emptyset, C_i, e)$ .
2. Extend  $\{A_1, A_2\}$  to a maximal available matching-pair at time  $i$ .
3. For each critical pair  $\{(u_1, v_1), (u_2, v_2)\}$  at time  $i$  for which  $\{A_1, A_2\}$  is weakly good, if  $\{A_1, A_2\}$  favors  $u_1$ , then color  $(u_1, v_1)$  green and color  $(u_2, v_2)$  black; otherwise, color  $(u_1, v_1)$  black and color  $(u_2, v_2)$  green.
4. Select an  $h \in \{1, 2\}$  uniformly at random.
5. Move the edges in  $A_h$  from  $C_i$  to  $M$ .

Figure 4. Processing  $C_i$  when there is no dangerous pair at time  $i$ .

When there is at least one dangerous pair at time  $i$ , the processing of  $C_i$  is very complicated and is omitted here for lack of space. What we can show is the following:

**Lemma 4.1** *Let  $S$  be the set of  $C_i$ -settled edges. Suppose that there is at least one dangerous pair at time  $i$ . Then, we can process  $C_i$  so that  $\mathcal{E}[w(S \cap M'_i)] \geq 11w(S)/80$ .*

## 5 The Main Result

Suppose that  $T$  is a maximum-weight tour of  $G$ . Let  $T_{\text{int}}$  denote the set of all edges  $\{u, v\}$  of  $T$  such that some cycle  $C$  in  $\mathcal{C}$  contains both  $u$  and  $v$ . Let  $T_{\text{ext}}$  denote the set of edges in  $T$  but not in  $T_{\text{int}}$ . Let  $\alpha = w(T_{\text{int}})/w(T)$ . By Lemmas 3.3 and 4.1, we can prove the following:

**Lemma 5.1** *Let  $\delta w(T)$  be the expected total weight of edges moved from  $\mathcal{C}$  to  $M$  at Step 4 or 5 in Figure 1. Then,  $\mathcal{E}[w(T_2)] \geq (0.5 + \delta)w(T)$  and  $\mathcal{E}[w(T_3)] \geq ((1 - \delta) + \frac{11}{160}(1 - \alpha))w(T)$ .*

Hassin and Rubinstein [4] show that  $w(T_1) \geq (1 - \epsilon)\alpha w(T)$ . So, we have:

**Theorem 5.2** *For any fixed  $\epsilon > 0$ , there is an  $O(n^3)$ -time approximation algorithm for Max TSP achieving an expected approximation ratio of  $\frac{251(1-\epsilon)}{331-320\epsilon}$ .*

## References

- [1] A. I. Barvinok, D. S. Johnson, G. J. Woeginger, and R. Woodroffe. Finding Maximum Length Tours under Polyhedral Norms. *IPCO'98*, LNCS, **1412** (1998) 195–201.
- [2] P. Chalasani and R. Motwani. Approximating Capacitated Routing and Delivery Problems. *SIAM Journal on Computing*, **28** (1999) 2133–2149.
- [3] R. Hassin and S. Rubinstein. An Approximation Algorithm for the Maximum Traveling Salesman Problem. *Information Processing Letters*, **67** (1998) 125–130.
- [4] R. Hassin and S. Rubinstein. Better Approximations for Max TSP. *Information Processing Letters*, **75** (2000) 181–186.
- [5] R. Hassin and S. Rubinstein. A 7/8-Approximation Approximations for Metric Max TSP. *Information Processing Letters*, **81** (2002) 247–251.
- [6] A. V. Kostochka and A. I. Serdyukov. Polynomial Algorithms with the Estimates  $\frac{3}{4}$  and  $\frac{5}{6}$  for the Traveling Salesman Problem of Maximum (in Russian). *Upravlyaemye Sistemy*, **26** (1985) 55–59.
- [7] A. I. Serdyukov. An Algorithm with an Estimate for the Traveling Salesman Problem of Maximum (in Russian). *Upravlyaemye Sistemy*, **25** (1984) 80–86.